

# GO-03 03: Модули и пакеты. Создание модулей и их версионирование

## Описание:

Мы научились использовать библиотеки из разных источников. Локальные пакеты, базовые библиотеки GO и библиотеки сторонних разработчиков из удаленных репозиториях - все эти источники пакетов чаще всего используются в любом проекте, написанном на Go. Будь это микросервис или любое другое приложение. Но также хорошей практикой является написание собственных модулей и библиотек. Если декомпозировать код приложений на модули, то это может положительно повлиять на скорость написания микросервисов и других приложений. В этом случае мы выделяем какую-то типовую логику в отдельный модуль, а затем используем этот модуль в разных приложениях и делаем это, используя `go mod`.

Поэтому предлагаю сейчас научиться создавать собственные модули и написать собственную публичную библиотеку, которую сможет использовать любой разработчик, пишущий на GO. Также научимся поддерживать эту библиотеку и заниматься ее версионированием.

Создание библиотеки

Давайте создадим пакет, который превратим в модуль и загрузим в публичный репозиторий на github.

В предыдущих заданиях мы создавали функцию, которая может проверять, содержит ли слайс строк определенную строку.

```
func Contains(a []string, x string) bool {
    for _, n := range a {
        if x == n {
            return true
        }
    }
    return false
}
```

Функция `Contains` как раз определяет, содержит ли слайс строк определенную строку. Это простая, но достаточно полезная функция, которая может пригодиться в любом приложении. Поэтому имеет смысл ее вынести в отдельный пакет, в котором у нас впоследствии будут находиться и другие подобные вспомогательные функции. Назовем этот пакет `utils`.

Итак, начнем с создания директории для нашего модуля. Создаем модуль не в `$GOPATH`, так как мы знаем из прошлого задания, что модули должны находиться вне этой директории.

```
~/mygo
├── gopackages
│   ├── main.go
│   ├── wordz
│   │   └── wordz.go
│   ├── color
│   │   └── color.go
│   └── utils
│       └── utils.go
```

Создаем внутри директории основной файл нашего пакета utils.go.

```
//файл utils.go
```

```
package utils

func Contains(a []string, x string) bool {
    for _, n := range a {
        if x == n {
            return true
        }
    }
    return false
}
```

Создаем reusable пакет utils. Добавляем в него функцию Contains. Не забываем, что функция должна быть экспортируемая (exported). Поэтому называем ее с большой буквы. На этом наш пакет готов.

Теперь нужно добавить нашу библиотеку в git-репозиторий и залить его на github.com.

```
$ git init
Initialized empty Git repository in /Users/dhnikolas/mygo/utils/.git/
```

```
$ git add utils.go
```

```
$ git commit -m "init commit"
[master (root-commit) a2245e9] init commit
1 file changed, 12 insertions(+)
```

```
create mode 100644 utils.go
```

Добавили пакет в Git и закоммитили все, что у нас есть. Теперь создаем пустой репозиторий на гитхаб. Называем его utils и делаем публичным. Ссылка на пустой репозиторий должна получиться такая <https://github.com/dhnikolas/utils.git> (в вашем случае будет другое имя пользователя).

Заливаем наш проект в этот репозиторий:

```
$ git remote add origin https://github.com/dhnikolas/utils.git
$ git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 339 bytes | 339.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/dhnikolas/utils.git
 * [new branch]      master -> master
```

Отлично! У нас есть собственная библиотека, опубликованная в публичном доступе. Теперь каждый разработчик может ее использовать. До появления go mod нужно было ее скачать утилитой go get github.com/dhnikolas/utils без привязки к версии и т.д. Но теперь так никто не делает, намного лучше и удобнее использовать не отдельный пакет, а модуль, с помощью которого мы будем релизить версии и автоматически подтягивать зависимости.

Пришло время превратить нашу библиотеку в модуль. Запустим команду go mod init github.com/dhnikolas/utils в директории с пакетом:

```
$ cd ~/mygo/utils/
$ go mod init github.com/dhnikolas/utils
go: creating new go.mod: module github.com/dhnikolas/utils
```

Тем самым мы создали файл go.mod и превратили пакет в модуль! Обратите внимание, что модуль обязательно должен называться так же, как путь импорта. То есть в нашем случае модуль должен называться github.com/dhnikolas/utils. Это обязательное условие go mod.

Далее закоммитим появившийся файл go.mod. После этого пакету нужно присвоить версию. Делается это через теги гита.

```
$ git add go.mod && git commit -m "go.mod"
$ git tag v1.0.0
$ git push origin master --tags
```

Присвоили текущему коммиту версию v1.0.0 и запустили в удаленный репозиторий. Это очень важный момент, так как версия на уровне go mod определяется как раз по этим тегам. Мы готовы использовать нашу новую библиотеку как стороннюю зависимость в проекте gopackages. Для этого давайте проимпортируем ее в файле main.go:

//файл ~/mygo/gopackges/main.go

```
package main

import (
    "fmt"
    "github.com/fatih/color"
    "gopackages/wordz"
    newcolor "gopackages/color"
    "gopackages/random"
    "github.com/dundoo/xstrings"

    "github.com/dhnikolas/utlis" //Импортируем нашу библиотеку как
    стороннюю зависимость
)

func main() {

    //С помощью нашей новой библиотеки проверяем слайс на наличие
    определенного значения
    //Если его там находим, то сообщим это в stdout и закончим
    выполнение программы
    isExist := utlis.Contains(wordz.Words, "Two")
    if isExist {
        fmt.Println("Slice Words contain finding value")
        return
    }

    newcolor.Greet()
    fmt.Println("Hello world")
    color.Red("Hello world again")

    fmt.Println(wordz.Hello)
    wordz.Words = []string{"Moscow", "New-York", "Amsterdam",
    "Barcelona", "Paris"}
    fmt.Println(wordz.Random())
}
```

```

wordz.Prefix = ""
fmt.Println(random.City())
fmt.Println(random.Digit())

fmt.Println(xstrings.Shuffle(random.City()))
fmt.Println(xstrings.Shuffle(random.Digit()))

}

```

Добавили импорт нашей библиотеки. Имейте в виду, что в вашем случае адрес импорта для библиотеки должен быть другой!

"github.com/{ваше\_имя\_пользователя\_на\_гитхабе}/utils" После этого вызываем функцию Contains из пакета utils, передаем для нее первым аргументом слайс строк, а вторым - искомую строку. В возвращаемом значении получаем булеву переменную isExist, которая говорит нам, есть ли такая строка в слайсе. Если есть, то сообщаем об этом и завершаем выполнение программы. Пробуем запустить.

```

$ go run main.go
go: finding module for package github.com/dhnikolas/utils
go: downloading github.com/dhnikolas/utils v1.0.0
go: found github.com/dhnikolas/utils in github.com/dhnikolas/utils
v1.0.0
Function init in package wordz
Slice Words contain finding value

```

Видим, что Go автоматически скачал версию v1.0.0 нашей библиотеки из гитхаба.

```

$ ls ~/go/pkg/mod/github.com/dhnikolas/utils@v1.0.0/
go.mod  utils.go

```

Это как раз то, что нам нужно. Мы создали библиотеку, залили ее в открытый доступ на гитхаб. И теперь любой разработчик может использовать ее. И с помощью go mod это делается очень просто!

В этот пакет мы будем продолжать добавлять полезные функции и дальше продолжим использовать и развивать эту библиотеку.

Минорная версия и патч

Давайте теперь обновим библиотеку и добавим в пакет utils еще одну функцию, которая будет делать то же самое, что и Contains, но для слайса integer-ов. Назовем ее ContainsInt.

```

//файл utils.go

```

```

package utils

func Contains(a []string, x string) bool {
    for _, n := range a {
        if x == n {
            return true
        }
    }
    return false
}

func ContainsInt(a []int, x int) bool {
    for _, n := range a {
        if x == n {
            return true
        }
    }
    return false
}

```

После этого закоммитим изменения и обновим тег с версией.

```
$ git add utils.go
```

```
$ git commit -m "Add new finctions ContainsInt"
[master 9c0c655] Add new finctions ContainsInt
1 file changed, 9 insertions(+)
```

```
$ git tag v1.1.0
```

```
$ git push origin master --tags
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 431 bytes | 431.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/dhnikolas/utils.git
* [new tag]          v1.1.0 -> v1.1.0
```

В библиотеках Go мы должны поддерживать [семантическое версионирование](#) - мажорную, минорную версии и патч.

- **МАЖОРНАЯ** версия - когда сделаны обратно несовместимые изменения.
- **МИНОРНАЯ** версия - когда вы добавляете новую функциональность, не нарушая обратной совместимости.
- **ПАТЧ**-версия - когда вы делаете обратно совместимые исправления.

В этот раз мы добавили новую функциональность, не нарушая обратную совместимость, поэтому инкрементировали минорную версию. Получилась v1.1.0. Также обратите внимание, что тег с версией имеет именно такой формат - с буквой "v" в начале - v1.1.0. Чтобы использовать новую версию библиотеки utils, в проекте gopackages нужно зайти в файл go.mod и изменить версию подключаемой библиотеки на 1.1.0.

файл ~/mygo/gopackages/go.mod

```
module gopackages
```

```
go 1.14
```

```
require (  
    github.com/dhnikolas/utils v1.1.0 // изменили версию тут  
    github.com/fatih/color v1.9.0  
    github.com/huandu/xstrings v1.2.1  
)
```

И в файле main.go вызовем вновь добавленную функцию.

```
package main
```

```
import (  
    "fmt"  
    "github.com/fatih/color"  
    "gopackages/wordz"  
    newcolor "gopackages/color"  
    "gopackages/random"  
    "github.com/huandu/xstrings"  
  
    "github.com/dhnikolas/utils" )
```

```
func main() {
```

```
//Вызвали новую функцию из пакета utils
isExistInt := utils.ContainsInt([]int{1,2,3,4,5}, 5)
if isExistInt {
    fmt.Println("Slice Int contain finding value")
    return
}

isExist := utils.Contains(wordz.Words, "Two")
if isExist {
    fmt.Println("Slice Words contain finding value")
    return
}
...
```

Запускаем

```
$ go run main.go
go: downloading github.com/dhnikolas/utils v1.1.0
Function init in package wordz
Slice Int contain finding value
```

Go скачал версию 1.1.0 нашей библиотеки, и новая функция отработала успешно! У нас получилось изменить минорную версию и сразу же начать ее использовать.

Мажорная версия

Изменения минорной версии и патча происходит по одному и тому же принципу. Чего не скажешь про изменение мажорной версии, так как в ней не сохраняется обратная совместимость на уровне самой библиотеки, но ее можно сохранить на уровне кода приложения, в котором эта библиотека используется. С точки зрения go mod разные мажорные версии приложения - это абсолютно разные пакеты. Рассмотрим этот важный момент поподробнее на примере.

Допустим, нам потребовалось внести в библиотеку utils обратно несовместимые изменения. А именно - переименовать функцию Contains в InSlice. Давайте это сделаем. Сперва нужно создать новую ветку в Git:

```
$ git checkout -b v2
Switched to a new branch 'v2'
```

Изменить название функции:

```
//файл utils.go
```

```
package utils
```



```
func InSlice(a []string, x string) bool {
    for _, n := range a {
        if x == n {
            return true
        }
    }
    return false
}
```

```
func ContainsInt(a []int, x int) bool {
    for _, n := range a {
        if x == n {
            return true
        }
    }
    return false
}
```

Изменить путь импорта в файле go.mod на github.com/dhnikolas/utils/v2:  
файл go.mod

```
module github.com/dhnikolas/utils/v2
```

```
go 1.14
```

Коммитим и добавляем новый тег к этому коммиту:

```
$ git add go.mod
```

```
$ git add utils.go
```

```
$ git commit -m "Change func name"
```

```
[v2 a7ed818] Change func name
```

```
2 files changed, 2 insertions(+), 2 deletions(-)
```

```
$ git tag v2.0.0
```

```
$ git push origin v2 --tags
```

```
Enumerating objects: 7, done.
```

```
Counting objects: 100% (7/7), done.
```

```
Delta compression using up to 12 threads
```

```
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 370 bytes | 370.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/dhnikolas/utils.git
* [new branch]      v2 -> v2
* [new tag]         v2.0.0 -> v2.0.0
```

Теперь у нас есть новая мажорная версия и нужно начать ее использовать. Так как это, по сути своей, совсем другой пакет, то мы можем использовать ее, не беспокоясь о нарушении обратной совместимости.

Импортируем версию v2 в пакет gopackages файл main.go.

```
utilsV2 "github.com/dhnikolas/utils/v2"
```

v2 - это другой пакет, но он имеет точно такое же название, как и пакет с первоначальной версией - utils. И если мы хотим использовать и старую, и новую версии библиотеки, то мы должны импортировать новую версию через псевдоним. Затем вызываем функцию InSlice: //файл main.go

```
package main
```

```
import (
    "fmt"
    "github.com/fatih/color"
    "gopackages/wordz"
    newcolor "gopackages/color"
    "gopackages/random"
    "github.com/dundxx/xstrings"

    "github.com/dhnikolas/utils"
    utilsV2 "github.com/dhnikolas/utils/v2"
)

func main() {

    isExistV2 := utilsV2.InSlice(wordz.Words, "Two")
    if isExistV2 {
        fmt.Println("Using utilsV2.InSlice and find value ")
        return
    }
}
```

```

isExistInt := utils.ContainsInt([]int{1,2,3,4,5}, 5)
if isExistInt {
    fmt.Println("Slice Int contain finding value")
    return
}

isExist := utils.Contains(wordz.Words, "Two")
if isExist {
    fmt.Println("Slice Words contain finding value")
    return
}

...

$ go run main.go
go: finding module for package github.com/dhnikolas/utils/v2
go: downloading github.com/dhnikolas/utils/v2 v2.0.0
go: found github.com/dhnikolas/utils/v2 in
github.com/dhnikolas/utils/v2 v2.0.0
Function init in package wordz
Using utilsV2.InSlice and find value

```

Go скачал еще одну версию библиотеки. И программа успешно работает, используя 2 пакета одной и той же библиотеки, благодаря импорту через псевдоним. К старой библиотеке мы обратились по ее оригинальному имени `utils`, а к новой версии - по псевдониму `utilsV2`. Такой вот гибкий и с первого взгляда неочевидный способ использует `go mod` для работы с библиотеками и их версионированием.

Подведем итоги!

- Для публикации собственной библиотеки нужен только любой онлайн-хостинг репозитория, например `github.com`.
- Управление версиями `go` модулей происходит с помощью тегов гита.
- В `go mod` необходимо использовать семантическое версионирование.
- Минорные версии и патчи находятся в одной ветке и в одном пакете. А мажорные должны использовать другую ветку в репозитории для обратной совместимости и импортироваться в проект как отдельные пакеты.

## Полезные ссылки:

- <https://habr.com/ru/post/421411/>
- <https://semver.org/lang/ru/>
- [https://golang.org/cmd/go/#hdr-Module\\_compatibility\\_and\\_semantic\\_versioning](https://golang.org/cmd/go/#hdr-Module_compatibility_and_semantic_versioning)

## Задание:

Дальше мы планируем использовать созданную нами сегодня библиотеку. Поэтому давайте ее немного улучшим.

1. Переименуйте функцию `ContainsInt` на `InSliceInt`.
2. Так как это изменение названия функции не обратно совместимо, примите необходимые меры по правильному версионированию библиотеки.
3. В ответе к заданию скиньте ссылку на итоговый репозиторий в github-e.